Evidence Flow Graph Methods for Validation and Verification of Expert Systems

Report IMP8809

Final Report on NASA Grant NAG-1-809

Lee A. Becker
Peter G. Green
Jayant Bhatnagar

Intelligent Machines Project
Artificial Intelligence Research Group
Worcester Polytechnic Institute
Worcester, Mass. 01609

Evidence Flow Graph Methods for Validation and Verification of Expert Systems

Lee A. Becker
Peter G. Green
Jayant Bhatnagar

ABSTRACT

This final report describes the results of an investigation into the use of evidence flow graph techniques for performing validation and verification of expert systems. This was approached by developing a translator to convert horn-clause rule bases into evidence flow graphs, a simulation program, and methods of analysis. These tools were then applied to a simple rule base which contained errors. It was found that the method was capable of identifying a variety of problems, for example that the order of presentation of input data or small changes in critical parameters could affect the output from a set of rules.

TABLE OF CONTENTS

Evidence Flow Graph Methods for Validation and Verification of Expert Systems

Lee A. Becker
Peter G. Green
Jayant Bhatnagar

1. INTRODUCTION

This final report describes the results of an investigation into the use of evidence flow graph techniques for performing validation and verification of expert systems. Simple expert systems consist of a set of rules, a data base, and an inference engine as shown in figure 1.

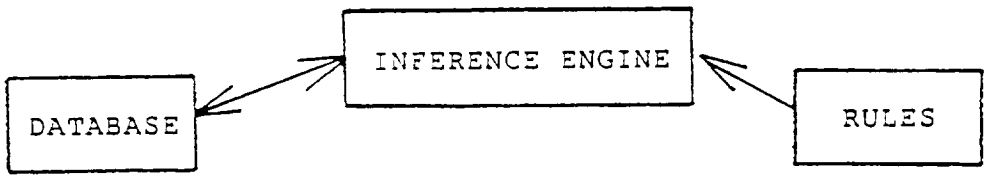
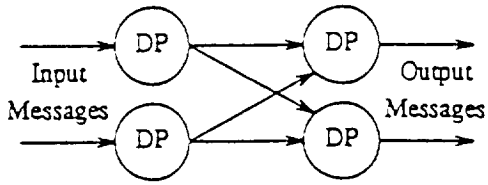


Figure 1: A Simple Expert System

An evidence flow graph representation (MIC87) for such a system replaces each rule with a process that is triggered by the arrival of data into the data base, either from an external source or from the execution of another rule process, as shown in figure 2. The result is a graph with data flow properties (ACK82) in which input data triggers rule executions until outputs are generated.

2a: An Evidence Flow Graph

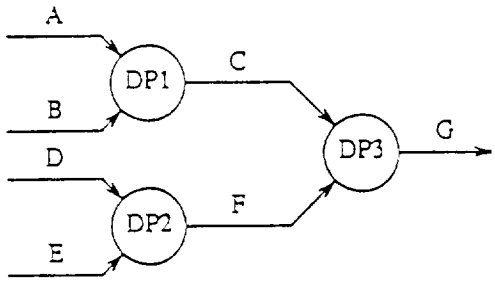


Decision process (DP) execution is triggered by the arrival of messages travelling along arcs.

2b: A Simple Rule-based Expert System

Rule 1: If A and B then C
Rule 2: If D and E then F
Rule 3: If C and F then G

2c: Rules mapped onto evidence flow graph



DP Definitions

DP1: When A and B arrive send C to DP3

DP2: When D and E arrive send F to DP3

DP3: When C and F arrive output G

Figure 2: Transformation of a Rule-Based Expert System into an Evidence Flow Graph

The potential importance of evidence flow graphs for V&V is twofold. First, the flow graph representation makes it possible to simulate the actions of the rules using discrete event simulation tools developed for analyzing complex systems. This opens up the possibility of performing monte-carlo performance tests under a wide variety of input values and timings. Second, the flow graph representation is independent of the inference engine and offers the potential for validating "portable" sets of rules which will work under any rule execution sequence imposed by an inference engine. This is also very significant when it is determined that correct operation is dependent on the inference engine, in which case the rules and inference engine must be validated and controlled as a unit.

The principal purpose of the research reported here was to determine whether evidence flow graph techniques would be potentially useful in aiding with the validation and verification of expert systems. This was approached by developing:

a) A translator to translate simple horn-clause rule bases into the evidence flow graph representation developed by Chisvin (CHI88) based on the work of Michalson (MIC88).

b) A simulation program written in SIMSCRIPT (RUS83,LAW84) to analyze the performance of the flow graph under a variety of conditions.

c) Methods for analysis which attempted to identify problems with rule execution by examining the output of the simulation program.

These tools were then applied to a simple rule base which contained errors and it was found that the method was capable of identifying problems, although it was evident that a much more sophisticated results analysis program will be needed if this technique is to be used on a large scale system.

It was found that the order of presentation of input data can affect the output from a set of rules. By corollary, the order in which the inference engine executes the rules may affect the output. This can cause problems when undesired outputs are produced before all the data is available or before all possible rules have been executed. It was also found possible to affect the resultant output by making small changes to critical parameters.

As a result of this investigation we have determined that evidence flow graph techniques can be used to find problems in rule based expert systems and that these techniques therefore have a place as part of the evaluation regime for the validation and verification of expert systems. Some of the faults found using evidence flow graph techniques, such as circular reasoning and unreachable conclusions, could be determined by other methods (SUW82,NGU85,NGU87,STA87,BEL87,JOH88). Some problems, such as critical sensitivity to parameters or the timing of data inputs, are uniquely suited to flow graph simulation techniques as is the determination of whether the rules are valid for any rule firing order.

To date experiments have been limited to simple horn-clause rule sets with most of the post simulation analyses being done by hand.

To make evidence flow graph techniques practically useful much work remains to be done. First, the work of Michalson (MIC88) needs to be extended to cover the translation of commonly used expert system shell paradigms into evidence flow graphs. Then some work needs to be done to build the software infrastructure to allow the automation of monte-carlo sensitivity and data timing simulations. Finally it is evident that an intelligent post processing program will be needed to find problems in the mass of data produced by the simulations. This program will probably be an expert system itself with knowledge about how to find faults from the results of the simulations.

This report presents a framework for validating expert systems in section 2. In section 3 the conversion of rules to evidence flow graphs is described followed by a description of the simulation program. Section 4 discusses the kinds of testing supported by the evidence flow graph approach and section 5 discusses the results, given in detail in the appendices, of the tests performed during this reseach. Finally in section 6, the report concludes with a summary of the results obtained to date and a favorable prognosis for the future use of these techniques in the validation and verification of expert systems.

2. A FRAMEWORK FOR VALIDATING EXPERT SYSTEMS

Figure 3 depicts our framework for the development of a validated expert system. One important feature of our approach is that the validation and verification is divided into a set of distinct processes. Performance analysis and verification takes place first

at the knowledge level, then again after information about the execution envirnoment has been incorporated. This is followed by a hardware failure effects analysis before testing in a simulated real-world environment. One cannot just verify a knowledge base. If the rule base is not invariant over all control strategies, then this must be known, and the rule base and control regime must be validated and verified as a pair. In addition, any modification to either the rule base or the control regime requires that the pair be revalidated. It is obvious that if the rule base were invariant over all control strategies, the control regime could be changed and revalidation would not be necessary. This would support portability.

Evidence Flow Graphs were developed at WPI as a representation for rule-based, Hearsay/Blackboard-based, and communication expert object-based expert systems (GRE87,MIC87). An evidence flow graph is a directed graph which represents decision making in terms of the collective behavior of several independent processes. The processes are characterized by the ability to make decisions in a limited problem domain and by the ability to communicate the results of these decisions by passing messages to other decision processes. The processes may range in complexity from simple

logical operations to implementations of complicated decision making paradigms. An evidence flow graph can execute in a manner similar to a data flow program. As messages arrive at a decision process they are stored until all the messages necessary for the decision process to execute are available. The process then "fires", consuming each input message, and generating new messages which are passed to those decision processes which require them.

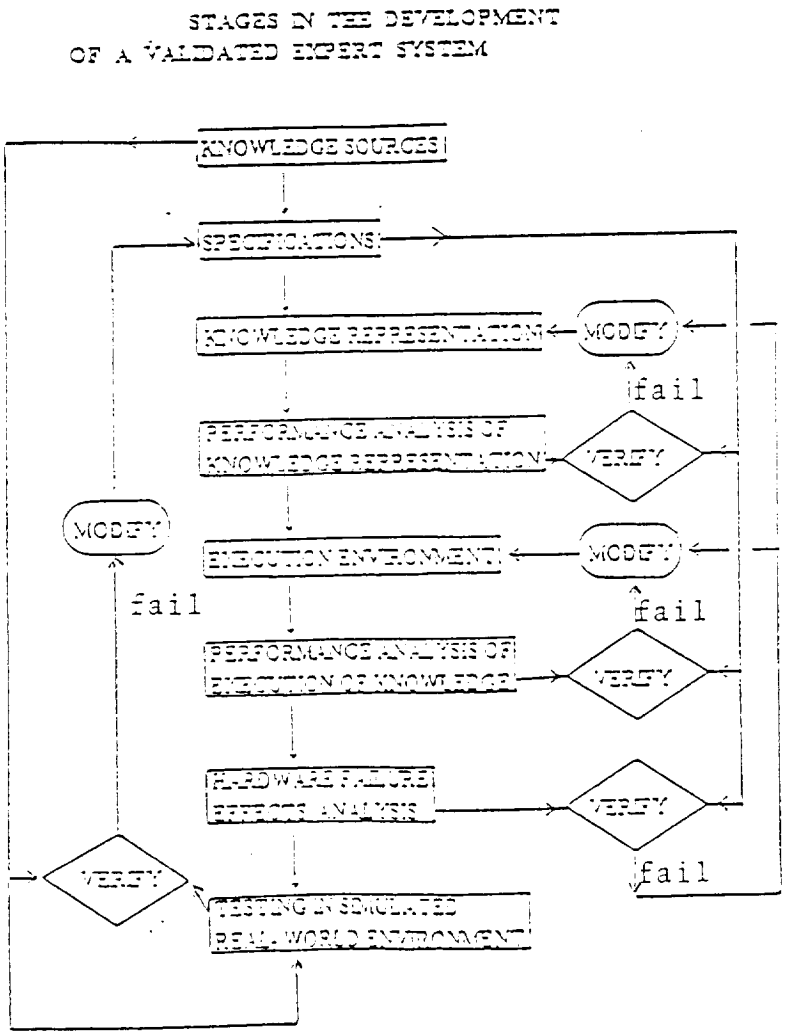


Figure 3: The Proposed Model for Validation of Expert Systems

The evidence flow graph thus provides a unified representation that can be mapped onto different computer hardware architectures. This is being investigated as part of an overall research project into how to build intelligent systems that are able to function in real-time in uncertain environments. These graphs are also of value in the validation and verification of expert systems.

Figure 4 depicts the use of flow graphs for performance analysis on different knowledge representations. An important feature of our approach is transforming the knowledge representation used into a graph theoretic form from which it can be analyzed and simulated using techniques developed for non-linear control systems.

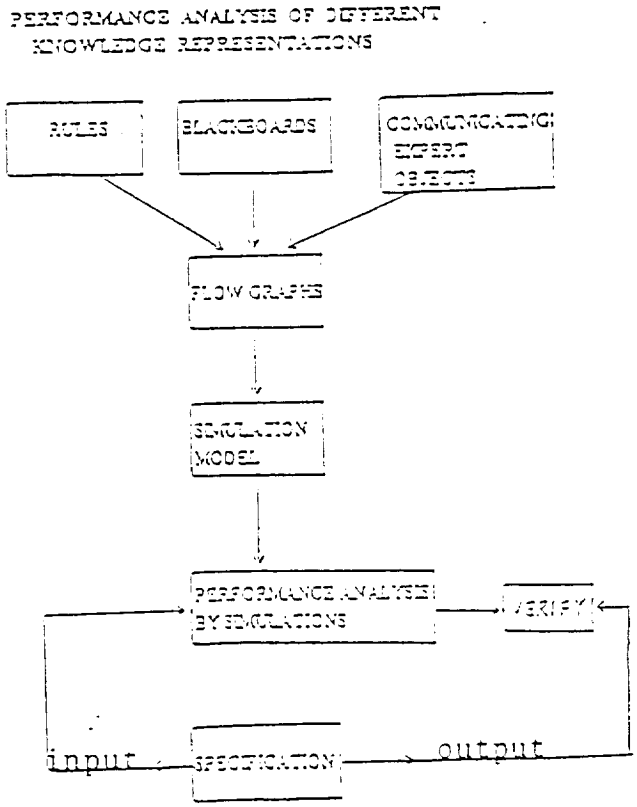


Figure 4: Evidence Flow Graphs as a Unifying Representation

3. KNOWLEDGE LEVEL VERIFICATION FOR A RULE-BASED REPRESENTATION

In this grant period we have investigated knowledge level verification. We have demonstrated our approach using a rule-based knowledge representation. Our system contains two modules: a rule-base to evidence flow graph translator and a simulation program. A third logical module, post-processing, currently is done by hand.

3.a. The Knowledge Representation to Evidence Flow Graph Translator

A translator takes the knowledge representation and yields an evidence flow graph. The knowledge is in the form of Horn-clause rules, where the antecedent is a conjunction of predicates and the consequent is a conclusion. There are specially designated input predicate nodes and output final conclusion nodes, as well as nodes for any subconclusions. For each rule there is a directed link from each of the predicates of the antecedent (input nodes or subconclusions) to the node of the conclusion or subconclusion in the consequent, as illustrated in Figure 5a. Weights on the links are based on the number of conjuncts. When a parameter is referred to in several relational predicates, there is a directed link from the parameter to each of the nodes for the relational predicates, as illustrated in Figure 5b.

A and B --> C

D and E --> F

F and G --> H

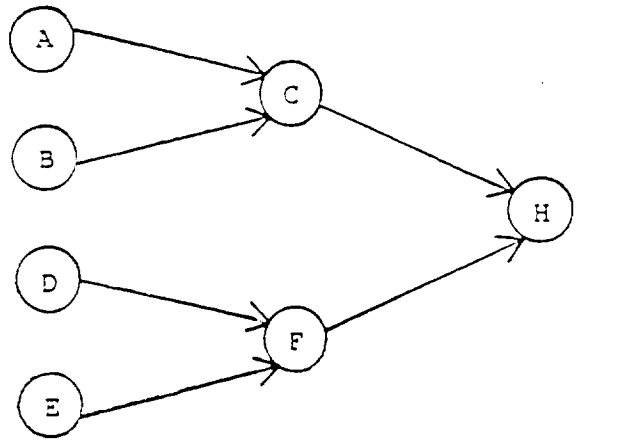


Figure 5a: Evidence Flow Graph Representation for Simple Rules

W > 200 and A --> B

W < 100 and C --> D

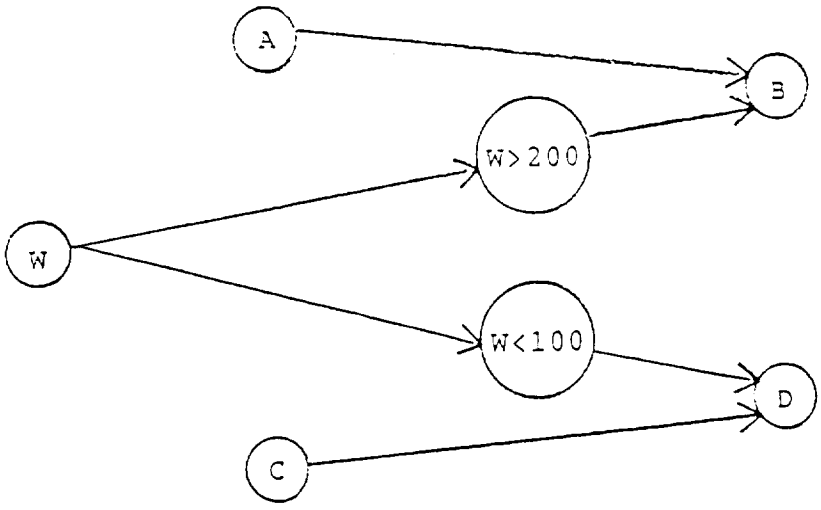


Figure 5b: Evidence Flow Graph Representation for Parameter Inputs

Conversion of a production rule base into an Evidence Flow Graph (EFG) is achieved by a translator implemented in LISP on a VAX/VMS 11/750 system.

The translator uses a depth-first strategy on the rule base to generate the nodes of the graph. Generation begins with the selection of an arbitrary rule from the rule base. Next, all rules leading to the same conclusion are collected to form a group. One conclusion node is created in the graph for each such group. For every conjunct of each rule in the group, a new node is created if one does not already exist. This node is treated as the conclusion node for a new group of rules that have the corresponding conjunct as their conclusion. Conjuncts that are specified as inputs to a rule and are mapped to the input nodes of the graph. Conjuncts which are not conclusions of any rule and which are not specified as inputs are treated as undefined and are flagged as errors. Graph generation continues until all conjuncts that appear in the rule base are mapped to the nodes of the graph.

For every conjunct that includes a logical comparison operator in its description, two nodes are established. One node is the value node that models an input node for the input parameter being compared, while the other is the comparison node that contains the threshold value against which the parameter value comparison is performed. A single arc connects the value node to the comparison node. In order for the comparison node to fire, a message must be received along this arc. All other conjuncts that perform a comparison of the same parameter against a different value have a different comparison node with an arc from the same input parameter

node.

The translator combines graph generation with static rule base checking that preceeds the dynamic testing implemented by the simulator (section 3.b.). Static checking enables the identification of those rules that contain undefined conjuncts in their conditions. In the event of detection of such a rule in the rule base, the translator logs an error in the error log file indicating the rule in error along with the conjunct that caused the error. From then on, the erroneous conjunct is treated as an input conjunct and graph generation continues as normal. On completion of graph generation, the translator issues a warning on the inconsistency of the generated graph arising from the assumed treatment of undefined conjuncts. Should no errors occur during graph generation/static checking, the translator outputs the graph in a canonical form which can then be modified for providing a formatted input to the simulator.

The reformatting of the translator output is provided to describe each node completely in the input to the simulator. Each node's description includes information on the type of node (e.g. input, output etc), the arc relations (e.g. conjunctive/disjunctive with respect to other arcs), and a description of each arc (e.g source node, relative importance of the arc for that node).

3.b. The Simulator

A simulator executes the evidence flow graph. All nodes, except for the input and relational predicate nodes, update with a weighted sum of the values of their input arcs. When several rules have the same conclusion, the update values are treated as a queue which takes the maximum of their input values. For example in Figure 6, E updates with the maximum of the weighted sums of A & B and of C & D if both are available, if only one sum is available it becomes the value of E, and if neither is available E will not be updated.

A and B --> E

C and D --> E

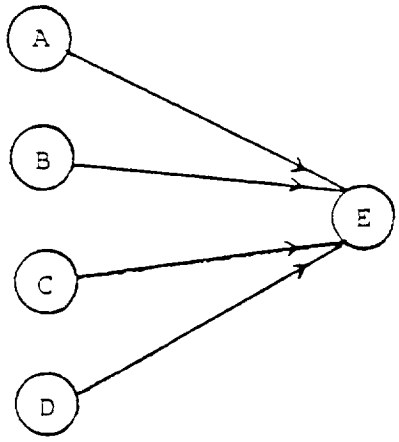


Figure 6: Several Rules with the Same Conclusion

The update values are sent as messages to nodes to which there is a directed arc. The values are real numbers between 0.0 and 1.0. For the initial stage of V & V (the knowledge level) it might be assumed that the work cells (nodes) fire as soon as their inputs are available and that there is no contention for computing

resources. One could then pick processing cell times at random from a distribution and test for many possible execution sequences. Simulation continues until all activity ceases in the network.

The motivation for using a 'universal' idealized, environment for the basis of knowledge level verification is portability and flexibility, perhaps also contributing to the possibility of hardware fault tolerance. The data flow-like processing allows one to consider the knowledge independently of the control strategy, and reflects inherent parallelism of expression of rules. It is also possible to verify a knowledge base under a particular control strategy. For example, a rule firing order mechanism for an inference engine, like a conflict resolution method, would be converted into a work cell scheduling mechanism for computing resources, in this case priorities of node firings.

The formatted output obtained from the translator provides input to the simulator, which is implemented in SIMSCRIPT II.5 on a VAX/ULTRIX 11/780 system.

Three major components comprising the simulator are :

(a) Decision Process Nodes

(b) Interconnection Arcs

(c) Communication Messages

Decision process nodes are centers of active decision making in the evidence flow graph. Broadly they may be classified into four main categories :

(a) Input nodes

(b) Output nodes

(c) Intermediate nodes

(d) Comparison nodes

Input nodes are entry points for symbolic and numeric information flowing into the graph from external environment. These nodes fire collectively in subsets as explained later in this section. Information leaving the input nodes appears as input to intermediate and/or comparison nodes.

Intermediate nodes are the centrally located nodes of the graph and are isolated from external environment by the input/output nodes. Input to intermediate nodes may appear from input, comparison, or other intermediate nodes. On collection of enough evidence at the intermediate node decisions regarding the subsequent flow of evidence are taken. Evidence flowing out from an intermediate node is either a confirmation or negation of the evidence arriving at its input.

Decisions at the graph nodes are conveyed to other nodes through flow of messages in the graph. No feedback information is made available at the input of graph nodes.

Comparison nodes are another type of nodes present in the evidence flow graph. Each comparison node has an arc from a parameter input node; they handle the flow of numeric evidence into the graph. Their functional description is provided later in this section.

All types of nodes have certain basic attributes like node-id, node-type, node-threshold, node-conclusion, and statistical counters that keep track of node activity in terms of the number of input messages received and the number of positive/non-positive

messages output by the node. Node-id of a node is its unique identification in the graph. Behavior of a node (i.e. action taken by a node on its activation) depends on the type of that node. Input nodes and intermediate nodes send messages to other nodes with activation values, while the output nodes produce the final conclusions. Node threshold is a static comparison value against which total evidence collected at a node is measured. If the evidence gathered at a node exceeds the firing the threshold, the node fires a boolean true value, else it fires with a boolean false value.

At any time during simulation, the nodes are either in an active state or in a state of hybernation. A message arriving from another node activates a hybernating node. Input arcs of nodes are checked for message arrivals. Message copies are deposited in input queues of the destination node/nodes specified in the message. During its activation, a node checks all relevant arcs for messages. If all conjunctive arcs have messages and at least one arc in the set of disjunctive arcs has a message, the node fires. Messages that initiate firing are removed from input queues and the node subsequently enters a hybernation state. Should the message requirements at the input arcs be insufficient, the node enters the hybernation state without firing. The activation sequence for nodes follows a fixed pattern - input nodes fire first, followed by the activation of comparison nodes, which in turn is followed by activation of all other types of nodes in the flow graph.

Associated with each node are entities called arcs. Incoming

messages are buffered in the arcs of a node. An arc may bear a conjunctive or disjunctive relationship to other incomming arcs of a node. As mentioned earlier, messages must be received along all conjunctive arcs and along at least one of the disjunctive arcs, for a node to fire. Each arc entity has attributes such as source node, a weight, a type (conjunctive/ disjunctive), and a count of the messages it receives. The weight of an arc is considered for determining the the importance of messages that are received along that arc. In the current system, in the translator the arc weight is computed by distributing the certainty factor of a node uniformly over the input arcs. Total evidence collected at a node is computed by taking the sum of products; each product is of the message activation level and the weight of the arc along which a message arrives. For all conjunctive arcs this value is summed, while for all disunctive arcs the maximum of values over all disjunctive arcs is selected.

Messages form the communication medium for inter-node communication. Each message is characterized a value (if the evidence it carries is a quantifiable numeric quantity), the weight of the corresponding arc, and a unique message number to uniquely identify the message in the system. Messages are consumed by a node on its firing. The output of nodes are messages that carry evidence representing a combination of evidence brought to a node by other messages plus the evidence generated at node itself.

The simulator operates in two phases: the setup phase and the simulation phase. During the setup phase, the description of each node of the graph is read from an input file and a corresponding

node is modelled as follows: if the node is an input node, a simulation entity modelling the node is created in the input nodes set. If the node is a non-input node, a process is created for modelling that node. The process description includes the node's attributes (e.g. its type), node entities and their attributes (e.g. arcs and their relation, relative weight etc.), and a procedure to simulate its action on activation. For each such process created, a process notice is placed in the future events set of the simulator, which works like a queue. For value-comparison nodes, two nodes are modelled in the simulator: The parameter value node is placed in the input nodes set, while the comparison node is associated with a process notice in the future events set. Process notices for all non-input nodes are scheduled to execute at the instant they are examined by the simulator. This immediate execution property of the co-routining node processes implements the inherent parallel exection model of the Evidence Flow Graph.

The simulation phase is made up of an arbitrary number of simulation cycles. Each simulation cycle corresponds to a selection of collective subsets of inputs in a way that allows inclusion of all input nodes in the set formed from the union of these subsets. Thus all input nodes are activated once in every simulation cycle in some permutation and combination of the inputs. At the start of the simulation phase, a random subset of inputs is generated and each input node is randomly assigned a boolean firing level. All process notices pending in the future events set are examined and their associated actions are executed.

The actions specified for intermediate nodes are as follows: verify that at least one message is queued in the buffer of every arc of a conjunction of input arcs and at least one message is queued in at least one of the arcs of every disjunctive set of input arcs. If all appropriate arcs have messages present in their buffers, compute the combined measure of evidence collected at this node. Check this measure against the node threshold, fire the node with a boolean value and delete all messages that contributed to current node firing. Firing of a node is equivalent to generating a new message and scheduling a corresponding process notice in the future events set with a priority of execution higher than the priority of execution of node process notices. Once a node has fired its execution is suspended. If, on the other hand, there are not enough messages received on appropriate arcs, then simply suspend execution of the node process.

The action sequence specified for output node processes is simpler. An output node qualifies for firing in the same way as an intermediate node does. If an output node qualifies for firing, then output the conclusion reached and suspend execution, else suspend the node process and continue with the current simulation cycle. During the exection of a message process notice mentioned in the action sequence outlined above, the action taken is to resume and reschedule all supended node processes, followed by storing messages in input buffers of arcs of every destination node in the future events set for which the message is intended.

On completion of a simulation cycle, the buffers of all arcs in the graph are cleared of any pending messages for a new subsequent

simulation cycle. Simulation cycles are repeated until the expiration of the user-specified duration of simulation.

3.c. The Post-Processor

The pattern of node firings (and message passings) is recorded in a logfile. A post-processor can then do various analyses on this file, for example to determine nodes that have never fired or nodes that have fired very often. It can be suggested that the rules corresponding to these nodes warrant additional scrutiny. The post-processor also can compile results from multiple runs with the same input, perhaps available at different times, so we can compare results from different input orderings and with different node firing orders to see if the results are always the same. In other words, the output of the post-processor will allow the identification of invariance of results with different input orderings, with different firing orders, as well as with parameter variation.

4. KINDS OF TESTING

A variety of different kinds of testing are supported by this approach. The most common type of checking done on expert systems is for consistency (SUW82,NGY85,NGY87,STA87). Static analysis on the evidence flow graph can yield this kind of information. In fact, several systems which do consistency checking translate a rule base into a inference net or graph for their analysis (STA87,BEL87). Such a graph structure could also be used to derive or generate sets of inputs for structural testing, if desired

(STA87). We concentrate here on the kinds of dynamic testing which can be done using simulation.

If there are available test cases which specify the conclusion to be reached for a set of inputs these can be run and wrong conclusions can be detected. This kind of testing can also be readily done by running the expert system itself, but there may be significant difficulty in assembling a large, well distributed set of test cases (OKE86). With the proposed method there are a number of kinds of testing which do NOT require the availability of test cases. All these involve running multiple simulations with randomly generated inputs values within the operational profile.

One type of testing which is very significant which does not require knowing the desired conclusion for a set of input values is testing whether the same conclusion will be reached regardless of the order that the input values become available. This is relevant when the system acts on the basis of the first conclusion reached. For a given set of input values multiple runs are made with different orderings of various subsets of the inputs.

For sensitivity testing the values of parameters are randomly varied within their operational profiles to determine whether any parameter is critical in its effect on the input, i.e. small changes in its value cause changes in the output. The effects of different degrees of belief of input predicates can also be examined. The evidence flow graph can be partitioned to allow this testing to be carried out on only the relevant subset of nodes.

For some applications it may be possible or necessary to specify

critical conclusions that are to be reached only under certain conditions or are not to be reached under certain conditions. These specifications can be tested using multiple simulations with randomly generated inputs. Here it is critical to partition the evidence flow graph to allow more exhaustive testing.

5. RESULTS OF THE PROJECT

At present a prototype rule base to evidence flow graph translator has been completed; this is written in LISP. Simulation programs to run an evidence flow graph with varying input values and ordering has been completed; this is written in SIMSCRIPT. A rule base for a small expert system has been identified. It has been translated into an evidence flow graph, and the simulation programs have been used to run the network. In addition, we have inserted errors into the sample rule base and demonstrated the kinds of errors that the proposed approach can detect. This is discussed below.

Appendix 1 is the sample rule base. Appendix 2 is the evidence flow graph representation that was generated from this rule base by the graph generator. Appendix 3 is the symbolic representation of this evidence flow graph. This evidence flow graph representation is now described in detail. All the INPUT_NODES except for AGE and LENGTH are predicates; their input values will be truth values (1 is true and 0 is false, and values between 1 and 0 indicate degree of belief). AGE and LENGTH are parameter inputs. They are input as real values, and the functions in their corresponding COMPARISON_NODES return true values (between 1.0 and 0.0).

The OTHER_NODES correspond to conclusions, i.e. RHS's of rules. The OTHER_NODES which begin with an * are final conclusions. In each OTHER_NODE following the node number, conclusion, and certainty factor, there is a list of nodes which correspond to the conjuncts on the LHS of the rule.

The nodes for the conjuncts may be either INPUT_NODES, COMPARISON_NODES or other OTHER_NODES. Following the number of the conjunct node there is a real number between 0.0 and 1.0. This stands for the 'weight' on the link from the conjunct's node to the conclusion node. The OTHER_NODES are updated with a weighted sum. The value of each conjunct node is multiplied by its weight and these products are added together.

In node 24 for 'mammal,' the second element in the conjunct list consists of two nodes; these two nodes correspond to the second conjunct in the rule for 'mammal' in EX.1. The second conjunct in the rule was 'animal', and there were two rules with 'animal' as a conclusion, as there are two OTHER_NODES (4 and 6) with 'animal' as a conclusion. These two nodes are connected to node 24 by an OR-connection. The value of an OR-connection used for updating is the maximum value of the nodes which have so far send values to the updating node. For this rule base there are no ELSE_NODES, which are created for if-then-else rules. There are also no NOT_NODES, which are used when a negated predicate is a conjunct in the LHS of a rule; the same predicate can thus be referred to positively and negatively in different rules.

The errors that can be detected by our techniques can be divided

into three classes:

1. sensitivity (over-sensitivity) of the conclusion reached to input parameters,
2. reaching different conclusions from one set of input values, and
3. errors which other researchers have detected using analytic methods but which may also be detected using our techniques.

Appendix 4 is a sample run with the input values on the right. The input values are randomly generated. For this run the conclusion 'ostrich' was reached. For sensitivity testing of the parameter inputs (class 1 above), the values of the other inputs may be held constant while just the parameters are varied. Alternatively a post-processor could take the results of the randomly generated input values, group together those which differ only in a parameter input, and perform analysis on the groups.

For a given set of rules, it possible that several conclusions can be reached from a single set of input values. This may be undesirable when an action is to be taken on the basis of the first conclusion reached and when the input values become available dynamically, i.e. not simultaneously. The simulation program also runs in a dynamic mode. This is illustrated in appendices 5, 6, 7. Appendices 6, 7, and 8 will be used to present an example of the second class of errors.

As appendix 5 shows, the rule base represented by the graph can be run with just a subset of the input values, and additional subsets until all the values have been input or a conclusion is reached. If simulation with a subset of input values reaches a conclusion, no additional subsets are run. This is what has occurred in appendices 6 and 7. The input values for these two runs are both

subsets of those in appendix 8. In the runs in appendices 6 and 7, different conclusions have been reached based on the order in which the input values in appendix 8 have become available. This is an example of the second class of errors.

Appendix 9 gives several rules which were added to the original rule base. These will be used to illustrate the capabilities of the system for identifying the third class of errors, those which other researchers have detected using analytic methods but which may also be detected using our techniques.

The last rule was identified as problematic during the translation process which generates the graph. This is an example of an error in which an antecedent conjunct in the LHS of some rule is neither an input, nor an conclusion of some other rule. The conjunct 'killshuman' is not parenthesized and therefore indicates an intermediate conclusion; however, there is no rule which has 'killshuman' as its RHS. Such an error might have many 'sources'. For example, the conjunct could have been misspelled in either the LHS or the RHS of some rule, or the rule which concludes this conjunct could have been omitted, or it may have been intended that this conjunct be an input.

Appendix 10 illustrates the data on message arrivals which is stored for each node. Appendix 11 is a frequency of node firings report for the graph generated from the rules in appendix 1 with the rule for 'human' replaced by the one in appendix 9 and the rule for 'man' from appendix 9 added. The nodes which never fired positively should be examined more fully. This does not

necessarily indicate a problem, at least for the number of runs done, but it points to situations which bear greater scrutiny. For example, the conclusions 'shark' and 'ape' were never reached, but there is nothing wrong with the rules that lead to them. On the other hand, the conclusions 'man' and 'human' from the first two additional rules in appendix 5 also never were reached, and closer scrutiny indicates that these rules were circular, each requiring the other conclusion to be reached. Another possible cause for a conclusion never being reached would be if the rules leading to it were directly contradictory. For example, if the same predicate were used positively in the LHS of an intermediate conclusion A, and negatively in the LHS of an intermediate conclusion B, and the rule with C in its RHS had A and B in its LHS. Conclusion C would also be unreachable if A and B referred to mutually exclusive comparison. Running multiple simulations identifies the rules that bear greater scrutiny.

Appendix 12 is a subset of the graph in appendix 3. The original graph was partitioned, and appendix 8 contains only the nodes corresponding to the rules that can be used to lead to the conclusion 'ape'. The smaller graphs created by partitioning allow more simulations to be run in a given amount of time. They can be used for conclusions of particular interest, perhaps those which are only to be reached under certain conditions.

6. CONCLUSIONS

It has been shown that Evidence Flow Graph methods can be used to detect errors and inconsistancies in expert systems. This has been

demonstrated experimentally by taking an existing rule base and converting it automatically to an evidence flow graph. This flow graph was then used as the input to a simulation program which predicted the performance of the expert system under a variety of conditions. Faults were detected during the translation process and as a result of simulation runs.

The techniques developed were general in nature and have a number of advantages over other techniques for detecting problems as part of the validation and verification process:

a) It provides a uniform representation for various knowledge representations and control strategies.

b) The evidence flow graph allows for analysis to recognize unused inputs and subconclusions, unreachable conclusions, disjoint and hence partitionable subgraphs, and relationships between inputs and outputs. It also provides a visually comprehensible representation in which many of these can be readily recognized.

c) It allows for simulation using techniques developed for non-linear stochastic systems.

d) It allows the consideration of different orders of input availability, and potentially for multiple data values for a single parameter.

e) It allows for sensitivity testing to determine where small changes in the values of input parameters

will result in different conclusions.

It was concluded that evidence flow graph techniques do have a role to play in performing sensitivity analyses as part of the validation and verification process for expert systems. More work, however, needs to be done to make these techniques practically useful. Some of the major future activities needed are:

1) The development of a program that will automatically develop simulation test sequences based on meta-knowledge about such items as possible ranges of input data and order of data availability.

2) The development of a program to automatically analyze the output data from the simulation runs and to detect problems. The simulation program generates a large volume of data when performing monte-carlo analyses which it is not practical to examine by hand. This post processing program will need to embody knowledge about fualts that could occur and how to detect them.

3) Further development of techniques to partition flow graphs so as to reduce the search space for faults.

4) Expansion of the translation program so as to be able to handle more complex knowledge forms and to translate these into evidence flow graphs.

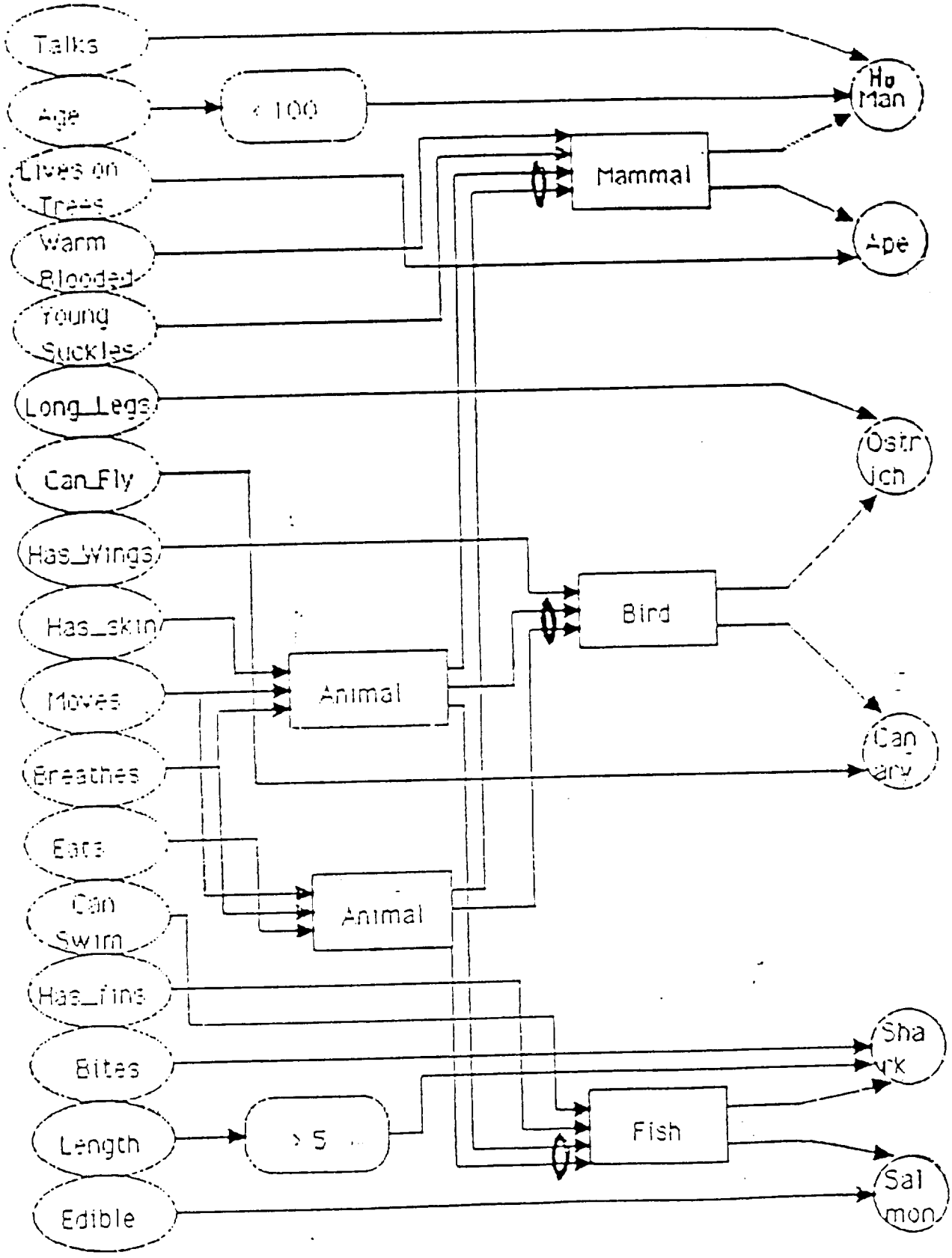
This past year we have made a successful start on techniques which can be used for the verification and validation of expert systems. The work described in this report has hopefully laid some of the foundation which can be used to assure that the expert systems used in our space program are reliable and safe.

Appendix 1: Sample Rule Base

LHS*	RHS	CF
(		
(((has_skin) (moves_around) (breathes))	animal	.9)
(((moves_around) (breathes) (eats))	animal	.9)
((animal (has_fins) (can_swim))	fish	.9)
(((bites) (length > 5) fish)	shark	.9)
(((edible) fish)	salmon	.9)
(((has_wings) animal)	bird	.9)
((bird (can_fly))	canary	.9)
((bird (long_legs))	ostrich	.9)
(((warm_blooded) animal (suckles_young))	mammal	.9)
((mammal (talks) (age < 100))	human	.9)
(((lives_on_trees) (age < 100) mammal)	ape	.9)
)		.

*The conjuncts are enclosed in parentheses, if they are input predicates, but not if they are inferred predicates, i.e. those on the RHS of some rule.

Appendix 2: Evidence Flow Graph Representation of the Rule Base



Appendix 3: Symbolic Representation of the Evidence Flow Graph

```
(
(INPUT_NODES
 (29  (LIVES_ON_TREES))
 (26  (AGE))
 (25  (TALKS))
 (23  (SUCKLES_YOUNG))
 (22  (LONG_LEGS))
 (18  (CAN_FLY))
 (16  (HAS_WINGS))
 (14  (EDIBLE))
 (11  (LENGTH))
 (10  (BITES))
 ( 8  (CAN_SWIM))
 ( 7  (HAS_FINS))
 ( 5  (EATS))
 ( 3  (BREATHES))
 ( 2  (MOVES_AROUND))
 ( 1  (HAS_SKIN))
)
(OTHER_NODES
 (* (30 APE      0.9  (((29  0.3))          ((27  0.3))      ((24  0.3)))))
 (* (28 HUMAN    0.9  (((24  0.3))          ((25  0.3))      ((27  0.3)))))
 (   (24 MAMMAL  0.9  (((22  0.3))          ((6  0.3)(4  0.3))    ((23  0.3)
 (* (21 OSTRICH 0.9  (((17  0.45))         ((20  0.45)))))
 (* (19 CANARY  0.9  (((17  0.45))         ((18  0.45)))))
 (   (17 BIRD    0.9  (((16  0.45))         (( 6  0.45)(4  0.45)))))
 (* (15 SALMON  0.9  (((14  0.45))         (( 9  0.45)))))
 (* (13 SHARK   0.9  (((10  0.3))          ((12  0.3))      (( 9  0.3)))))
 (   ( 9 FISH    0.9  (((6  0.3)(4  0.3)) ((7  0.3))       ((8 0.3)))))
 (   ( 6 ANIMAL  0.9  (((2  0.3)) ((3  0.3))       ((5  0.3)))))
 (   ( 4 ANIMAL  0.9  (((1  0.3)) ((2  0.3))       ((3  0.3)))))
)
(COMPARISON_NODES
(         (27  (AGE <)  1        (((26 1)))))
(         (12  (LENGTH >)  1     (((11 1)))))
)
(ELSE_NODES)
(NOT_NODES)
)
```

Appendix 4: Sample Run 1

NODE		VALUE	FIRED
Initial node firings:			
25	talks		0
22	warm_blooded		0
20	long_legs		1
18	can_fly		0
16	has_wings		1
5	eats		1
3	breathes		1
29	lives_on_trees		0
26	age	46.38	1
23	suckles_young		1
14	edible		0
11	length	4.90	1
10	bites		0
8	can_swim		1
7	has_fins		1
1	has_skin		0
2	moves_around		1

Conclusion:

THERE IS ENOUGH EVIDENCE (0.90) TO SUGGEST THAT ostrich

Appendix 5: Sample Run 2

NODE		VALUE	FIRED
Initial node firings:			
26	age	198.56	1
25	talks		0
20	long_legs		1
18	can_fly		0
23	suckles_young		0
14	edible		0
11	length	6.93	1
8	can_swim		1
29	lives_on_trees		1
Additional Node Firings:			
22	warm_blooded		0
7	has_fins		1
5	eats		1
1	has_skin		1
Additional Node Firings:			
16	has_wings		1
2	moves_around		1
Additional Node Firings:			
10	bites		0
3	breathes		1
Additional Node Firings:			
Input nodes exhaused			

>>>>>>>INSUFFICIENT EVIDENCE FOR REACHING ANY CONCLUSION<<<<<<<

Appendix 6: Sample Run 4

NODE		VALUE	FIRED
Initial node firings:			
1	has_skin		0
2	moves_around		1
25	talks		0
22	warm_blooded		0
11	length	4.90	1
20	long_legs		1
16	has_wings		1
5	eats		1
3	breathes		1
29	lives_on_trees		1

Conclusion:

THERE IS ENOUGH EVIDENCE (0.90) TO SUGGEST THAT ostrich

Appendix 7: Sample Run 5

NODE		VALUE	FIRED
Initial node firings:			
10	bites		0
8	can_swim		0
22	warm_blooded		0
18	can_fly		1
16	has_wings		1
2	moves_around		1
5	eats		1
3	breathes		1
26	age	46.38	1
23	suckles_young		1

Conclusion:

THERE IS ENOUGH EVIDENCE (0.90) TO SUGGEST THAT canary

Appendix 8: Sample Run 3

NODE		VALUE	FIRED
Initial node firings:			
25	talks		0
22	warm_blooded		0
20	long_legs		1
18	can_fly		1
16	has_wings		1
5	eats		1
3	breathes		1
29	lives_on_trees		1
26	age	46.38	1
23	suckles_young		1
14	edible		0
11	length	4.90	1
10	bites		0
8	can_swim		0
7	has_fins		0
1	has_skin		0
2	moves_around		1

Conclusion:

THERE IS ENOUGH EVIDENCE (0.90) TO SUGGEST THAT ostrich

Conclusion:

THERE IS ENOUGH EVIDENCE (0.90) TO SUGGEST THAT canary

Appendix 9: Some Additional Rules

(((has_beard) human)	man	.99)
((man (eats) (can_sing))	human	.99)
((ape kills_humans (moves_around))	monster	.9)

Appendix 10: DETAILS OF MESSAGE ARRIVALS ON NODE 9

NODE CONCLUSION : fish

SOURCE NODE	SOURCE CONCLUSION	ARC TYPE	NUMBER MESSAGES
6	animal	or	201
4	animal	or	201
7	has_fins	and	196
8	can_swim	and	201

Appendix 11: Frequency of Node Firings Report

NODE#	NODE CONCLUSION	ZERO	NON-ZERO	TOTAL
33	lives_on_trees	46	50	96
31	can_sing	48	48	96
29	has_beard	57	36	93
26	age	52	43	95
25	talks	46	50	96
23	suckles_young	52	42	94
22	warm-blooded	42	54	96
20	long_legs	45	51	96
18	can_fly	53	43	96
16	has_wings	51	45	96
14	edible	40	55	95
11	length	46	47	93
10	bites	48	48	96
8	can_swim	42	54	96
7	has_fins	51	45	96
5	eats	50	47	97
3	breathes	44	53	97
2	moves_around	47	50	97
1	has_skin	48	46	94
27	age < 100	73	22	95
12	length > 5	65	28	93
30	man	92	0	92
28	human	93	0	92
24	mammal	91	2	93
17	bird	87	9	96
9	fish	93	2	95
6	animal	82	15	97
4	animal	82	12	94
34	ape	92	0	92
21	ostrich	91	5	96
19	canary	90	5	95
15	salmon	93	2	95
13	shark	92	0	92

Appendix 12: Partitioned Graph for Output 'APE'

```
(

(INPUT_NODES

 (23  (SUCKLES_YOUNG))
 ( 1  (HAS_SKIN))
 ( 5  (EATS))
 ( 3  (BREATHES))
 ( 2  (MOVES_AROUND))
 (22  (LONG_LEGS))
 (26  (AGE))
 (29  (LIVES_ON_TREES))
)

(OTHER_NODES

 (* (30   APE     0.9  (((29  0.3))       ((27  0.3))      ((24  0.3)))))
 (   ( 4  ANIMAL  0.9  (((1  0.3))        ((2  0.3))       ((3  0.3)))))
 (   ( 6  ANIMAL  0.9  (((2  0.3))        ((3  0.3))       ((5  0.3)))))
 (   (24  MAMMAL  0.9  (((22  0.3)) ((6  0.3)(4  0.3)) ((23  0.3)))))
)

(COMPARISON_NODES

(        (27  (AGE <)  1 (((26 1)))))
)

(ELSE_NODES)

(NOT_NODES)

)
```

References

Ackerman, W.B. 1982. Data Flow Languages. IEEE Computer 15.2: 15-25.

Bellman, K. and Walter, D.O. 1987. Testing Rule-based Expert Systems. (Personal Communcation).

Chisvin, L. 1988. Using Discrete Event Simulation to Predict the Network Communication Performance of Message-Based Data Flow Multiprocessor Systems. Master's Thesis, Worcester Polytechnic Institute, Worcester, MA.

Green, P.G. and W.R. Michalson. 1987. Real-Time Evidential Reasoning and Network Based Processing. Proceedings of the IEEE First Annual International Conference on Neural Networks, Vol. 2, pp. 359-365.

Johnson, S. 1988. Validation of Highly Reliable, Real-Time Knowledge-Based Systems. Proceedings of the 2nd Annual Workshop on Space Operations Automation and Robotics (SOAR 88).

Michalson, W.R., Green, P.E., Duckworth, R.J. 1987. Evidence Flow Graphs: A Unified Representation for Distributed Artificial Intelligence Systems. Worcester Polytechnic Institute Report EE87IMP10.

Michalson, W.R. 1988. A Computing Architecture for Real-Time Decision Making. Ph.D. Dissertation, Worcester Polytechnic Institute, Worcester, MA.

Nguyen, T.A., Perkins, W.A., Laffey, T.J. and Pecora, D. 1985. Checking an expert system knowledge base for consistency and completeness. IJCAI9, pp. 376-378.

Nguyen, T.A., Perkins, W.A., Laffey, T.J. and Pecora, D. 1987. Knowledge base verification. AI Magazine, Vol.8, No.2., pp. 65-79.

Law, A.M. and Larmey, C.S. 1984. An Introduction to Simulation Using SIMSCRIPT II.5. O'Keefe, R.M., Balci, O., and Smith, E.P. 1987. Validating Expert System Performance. IEEE Expert, Vol.2, No.4., pp. 81-89.

Russell, E.C. 1983. Building Models with Simscript II.5. CACI, Inc., Los Angeles, CA.

Stachowitz, R.A., Chang, C.L., Stock, T.S., and Combs, J.B. 1987. Building Validation Tools for Knowledge-Based Systems. Proceedings of the First Annual Workshop on Space Operations Automation and Robotics (SOAR '87), pp. 209-215, NASA Conference Publication 2491, Houston, TX., August 1987.

Suwa, M., Scott, A.C. and Shortliffe, E.H. 1982. An approach to verifying completeness and consistency in a rule-based expert system. AI Magazine, Vol.3, No.4, pp. 16-21.